

# Touché Task 1: Argument Retrieval for Controversial Questions

Manuel Barusco<sup>1</sup>, Gabriele Del Fiume<sup>2</sup>, Riccardo Forzan<sup>3</sup>, Mario Giovanni Peloso<sup>4</sup>, Nicola Rizzetto<sup>5</sup> and Elham Soleymani<sup>6</sup>

<sup>1</sup>University of Padua, Italy

## Abstract

This document demonstrates the work of the LGTM Team of the University of Padua's participation in CLEF 2022 Touché Task 1, creating a system to retrieve arguments for controversial questions. This system aims to provide a solution to assist users who search for arguments to be used in conversations and retrieves a pair of sentences from a collection of arguments. In the various runs, we used a custom parser for parsing the document collection, a custom indexer, the BM25 Similarity, the DataMuse API for query expansion and boosting in Searcher. We also tried to improve the system by using different stoplists, stemmers and apply re-ranking based on sentiment analysis and readability of the document.

## Keywords

Information retrieval, Search engines, Lgtm-Team

## 1. Introduction

During the last decade, relentless optimization of information retrieval effectiveness has driven search engines to new quality levels. Most people are satisfied most of the time, and web search has become a standard and often preferred source of information finding.

Nevertheless, in recent years, a principal driver of innovation has been the World Wide Web, unleashing publication at the scale of tens of millions of content creators. This explosion of published information would be questionable if the information could not be found, annotated, and analyzed so that each user can quickly find relevant and comprehensive information for their needs. [1]

Moreover, One of the most critical parts of these search engines that can be improved is argument retrieval for controversial questions. To contribute to enhancing different solutions for this problem, we decided to participate in the Touché 2022 Lab proposed by CLEF[2]. We chose task 1 among three different tasks in the Touché lab to work with. This task aims to retrieve and rank sentences that carry significant points pertinent to the controversial topic.

The dataset used in this project is a pre-processed version of args.me dataset. For the complete development, we decided to use the downloadable version of the dataset. To solve this task,


---

*"Search Engines", course at the master degree in "Computer Engineering", Department of Information Engineering, and at the master degree in "Data Science", Department of Mathematics "Tullio Levi-Civita", University of Padua, Italy. Academic Year 2021/2022*

✉ manuel.barusco@studenti.unipd.it (M. Barusco); gabriele.delfiume@studenti.unipd.it (G. D. Fiume); riccardo.forzan@studenti.unipd.it (R. Forzan); mariogiovanni.peloso@studenti.unipd.it (M. G. Peloso); nicola.rizzetto.2@studenti.unipd.it (N. Rizzetto); elham.soleymani@studenti.unipd.it (E. Soleymani)



© 2022 Copyright for this paper by its authors.  
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

we used Lucene, a Java library providing powerful indexing and search features and advanced analysis and tokenization capabilities.

The organization of this paper contains these sections: Section 2 indicates the previously related work, Section 3 illustrates our approach and method to show a new solution; Section 4 denotes our experimental setup; Section 5 defines our solutions and results we had; ultimately, Section 6 implies the conclusions and outlooks for future work that might be possible to develop the result.

## **2. Related Work**

There have been various researches to solve the problem related to argument retrieval previously. Moreover, our initial step in developing an Argument retrieval system was the review of the previous year's[3] editions of the Touché Lab and the Hello Tipster repository provided by Professor Nicola Ferro.

There were some approaches and tools used in these cases, for instance: Using the Lucene implementation of BM25 and LMDirichlet Model to rank the relevancy of each document, changing the weight method in every field in documents. Furthermore, using synonyms from different sources in query expansion to improve the result and using the OpenNLP Machine learning toolkit to boost the tokenization procedure.

While, many of these approaches lead to an acceptable and usable system, some of them did not have a great functionality. In this section we would love to explain about some of the previous ideas that helped us to have a better vision of the ways we can improve an information retrieval system. We divided these approaches in two categories. The first category is the frequently used methods which are the common methods that have been used over these years although they have some drawbacks as well as their benefits. Additionally, Some unique approaches, that were not common in every study and the systems creators were innovative with a newborn methodology.

### **2.1. Frequent methods**

These tools and approaches have been used in most of the studies from last year. Some tools and approaches that were common in most of the projects are:

- Apache Lucene: An Information retrieval library which is written in Java [4]
- BM25, LMDirichlet, and Boolean model: The usage of the Lucene implementation of these model for Similarity.[5]
- Analysis of Variance (ANOVA): To test if survey or experiment results are useful or compare more than two groups simultaneously to determine whether a relationship exists between them.[6]
- PorterStemFilter and LovinsStemmer: Testing a variety range of Stemmers for instance KStemFilter, PorterStemFilter, and LovinsStemmer.
- Trec eval: A standard tool used by the TREC community for evaluating an ad hoc retrieval run, given the results file and a standard set of judged results.[7]

- Synonyms extraction from WordNet: Queries expanded by synonyms extracted from WordNet, an extensive lexical database of English nouns, verbs, adjectives, and adverbs, are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept.[8]

## 2.2. Unique methods

Some approaches and tools are unique in these cases, for instance: Changing the weight method in every field in documents. Furthermore, using the OpenAI GPT-2 algorithm, the OpenNLP Machine learning toolkit to boost the tokenization procedure, and the score obtained by sentiment analysis on the documents to re-rank them. However, not all of these tools resulted in an spectacular system. Although some of them did not have an remarkable result, they thought us to try or not to try new tools and ways to create a new system. That is the reason that we listed them here.

A list of these exclusive tools and a detailed description of them is shown below:

- EnglishMinimalStemmer (an updated version of the Porter stemmer) in the analyzer[9]
- LingPipe: a tool kit for processing text using computational linguistics, for instance, finding the names of people, organizations, or locations in the news or Suggesting correct spellings of queries.[10]
- The Apache OpenNLP library: a machine learning-based toolkit for processing natural language text which supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and coreference resolution. These tasks are usually required to build more advanced text processing services. OpenNLP also includes maximum entropy and perceptron-based machine learning. According to previous studies, it should be the best in accuracy terms, but it was very slow to index.[11]
- OpenAI GPT-2 model: a Machine Learning algorithm that generates synthetic text samples from arbitrary input. It is used in query expansion, giving the topic title input to develop a complete phrase with new words that could help the searching part. The output of GPT-2 was not as good as expected.
- Sentiment analysis: Re-ranking regards the score obtained by performing sentiment analysis on the documents using VADER (Valence Aware Dictionary and Sentiment Reasoner), a lexicon and rule-based sentiment analysis tool specifically attuned to sentiments expressed in social media.[12]

## 2.3. Our Approach

Our work evolved from the Hello Tipster [13] repository and several methods used by other studies as mentioned above that had marvelous results. Additionally, some of the ways of other studies lead to not satisfying results, so we learnt from them not to use those methods to optimize our approach.

We tried to develop the project to showcase our approach to a new information retrieval system. Several classes, methods, and filters are added to improve performance in indexing time, parsing, and searching. The details of the form of our solution are explained in the next section.

### 3. Methodology

We create our system starting from the standard pipeline of every Information Retrieval system: we develop the main parts for the parse, index, and search phases separately. Numerous methods and tools such as different filters, similarities, APIs, and search optimization techniques are used in this project to optimize the indexing, parsing, and searching of the documents. Other studies inspired us after we saw their precious results. So we used some of their primary, frequent, and unique tools. Some of these tools are:

- LengthFilter and EnglishPossessiveFilter in the analyzer
- Sentiment analysis in re-ranking
- BM25 and LMDirichletSimilarity similarities

After trying different tools and solutions in every part of the project, we tried to develop some methods that were not used in previous years' studies to improve the Information retrieval procedure. So to examine if these methods and tools are influential and they can enhance the pace of the system or lead us to an optimized system, we used these particular tools:

- Jackson library to process the collection
- Rapid Automatic Keyword Extraction (RAKE) to improve the query expansion process
- DataMuse API for searching adequate synonyms in query expansion
- Re-rank techniques based on the sentiment and readability analysis

Moreover, the sections below are indicated to demonstrate a better understanding of the workflow of this system. These sections provide every package in our source code with its objectives.

#### 3.1. Parsing, Pre-Processing, and Indexing

We developed a custom parser for parsing the collection itself for the parsing phase of the system pipeline. With this particular parser, we managed to parse the collection in the most meaningful way for our purpose. For this phase, two libraries are used:

- Jackson library: Jackson is a suite of data-processing tools for Java (and the JVM platform), including the flagship streaming JSON parser/generator library, matching data-binding library (POJOs to and from JSON), and additional data format modules to process data encoded in CSV, XML, and many other formats. We used this library for parsing some specific fields of the documents in the collection, which were encoded in a JSON format. [14]
- Apache CSV library for parsing the main CSV structure of the collection file.

Our system stores different information independently and with a proper weight in fields for creating Lucene Documents<sup>1</sup>. We decided to extract from every document in the collection the following fields:

---

<sup>1</sup>[https://lucene.apache.org/core/9\\_1\\_0/core/org/apache/lucene/document/Document.html](https://lucene.apache.org/core/9_1_0/core/org/apache/lucene/document/Document.html)

- ID: the id of the document
- Conclusion: the conclusion of the document
- Stance: the stance of the document (PRO or CON)
- AcquisitionTime: time of acquisition of the argument
- discussionTitle: title of the discussion in the debate website
- sourceTitle: title of the source debate page
- url: url of the argument
- sourceText: text of the argument
- sentences: key sentences of the argument

Then, after the extraction of these fields, we decided to index only the ID, CONCLUSION, DISCUSSION TITLE, SOURCE TITLE, and SOURCE TEXT fields. In our project, we indexed the collection documents by using Lucene default fields, and in order to give a better organization for our scope, we added some custom index fields:

- Bodyfield: A custom field for indexing the body of a document by storing the tokens, the term vectors, documents, frequencies, positions and offsets of the field. The whole content of the body is not stored.
- IdKeyField: A custom field for indexing the ID of a sentence or a document, or every field containing key values such as acquisition time, stance, etc. This field is not tokenized, and it is stored.
- BodyCorrelatedField: A custom field for indexing the additional information of the body field such as conclusion, stance, discussion title, source title, etc. The indexing options of this field are the same as the body field, but the field is stored in this case.

We decided to adopt the ClassicTokenizer<sup>2</sup> provided by Apache Lucene. It is a grammar-based tokenizer that is good for most European-language documents. This tokenizer splits the text field into tokens, treating whitespace and punctuation as delimiters. These delimiters are discarded with the following expectations:

- Periods (dots) that are not followed by whitespace are kept as part of the token.
- Words are split at hyphens unless there is a number in the word, in which case the token is not split and the numbers and hyphen(s) are preserved.
- Recognizes Internet domain names and email addresses and preserves them as a single token.

Beyond this tokenizer, we decided to use the LowerCaseFilter<sup>3</sup>, in order to normalize all tokens to lower case. This also allows terms of the query to match with terms in the documents written, for example in upper case. The next filter used is the EnglishPossessiveFilter<sup>4</sup> that

---

<sup>2</sup>[https://lucene.apache.org/core/6\\_5\\_0/analyzers-common/org/apache/lucene/analysis/standard/ClassicTokenizer.html](https://lucene.apache.org/core/6_5_0/analyzers-common/org/apache/lucene/analysis/standard/ClassicTokenizer.html)

<sup>3</sup>[https://lucene.apache.org/core/8\\_8\\_1/analyzers-common/org/apache/lucene/analysis/core/LowerCaseFilter.html](https://lucene.apache.org/core/8_8_1/analyzers-common/org/apache/lucene/analysis/core/LowerCaseFilter.html)

<sup>4</sup>[https://lucene.apache.org/core/7\\_3\\_1/analyzers-common/org/apache/lucene/analysis/en/EnglishPossessiveFilter.html](https://lucene.apache.org/core/7_3_1/analyzers-common/org/apache/lucene/analysis/en/EnglishPossessiveFilter.html)

removes possessives (trailing 's) from words. The last filter we used is LengthFilter<sup>5</sup>. This filter keeps tokens with a length between 3 and 20 characters, removing the others. Previous studies showed that this filter worked well in this collection. In the next section we talk about the usage of different stoplists and stemmers for system performance improvement. All the tests were done in a simplified version of the system where the search wasn't boosted and without any query expansion or re-rank technique. This was done in order to see the impact of that solution (stoplists or stemmer application) on the system. The similarity used was BM25.

### 3.1.1. Tested Stoplists

Stopword filtering is a common step in preprocessing text because it removes lots of not informative words. We tested different stoplists in the analyzer, and we realized that stoplists have a considerable impact on the system performances, so we tried different lists, as reported in Tab. 1 and we decided to keep the stoplist solution that gives us the best performance measures.

StopList	nDCG@5	P_5	RECALL
CoreNLP	0.4240	0.3519	0.8507
CountWordsFree	0.4240	0.3482	0.8496
EBSCO	0.4200	0.3522	0.8496
GoogleStop	0.4240	0.3519	0.8500
Ranks	0.4080	0.3362	0.8499
NO STOP	0.4200	0.3532	0.8524

Tab. 1: System performances with different stoplists

The different stop lists used are:

- CoreNLP: stoplist used in CoreNLP java library. This list contains 275 tokens.
- CountWordsFree: a list of stop words that are frequently used in English. It contains 851 words.
- EBSCO: list of 24 words used in EBSCOhost medical databases MEDLINE and CINAHL. We saw that this list gave good performances to previous years' projects.
- GoogleStop: simple English stoplist used by Google. It contains 174 words.
- Simple English stopwords list downloaded from ranks.hl . It contains 174 words.

The results in Tab. 1 show that the two better solutions are: NO STOP LIST and CoreNLP stoplists. But, after testing these two solutions in the final system, we saw that the no stop list solution is better. Since we don't have any index storage limitation, don't apply any stoplist isn't a problem.

---

<sup>5</sup>[https://lucene.apache.org/core/8\\_8\\_1/analyzers-common/org/apache/lucene/analysis/miscellaneous/LengthFilter.html](https://lucene.apache.org/core/8_8_1/analyzers-common/org/apache/lucene/analysis/miscellaneous/LengthFilter.html)

### 3.1.2. Tested Stemmers

After setting the best stoplist solution, we tried different stemmers. Stemming is the reduction of a word into its base form, called stem. In particular, we tried three different stemmers synthesized in Tab. 2. The stemmers can generalize the search process, but they can also worsen it, so we decided to test different stemmers and see which one gives us the best performance.

Stemmer	nDCG@5	P_5	RECALL
EnglishMinimalStemmer	0.3532	0.4200	0.8534
KStemFilter	0.3833	0.4300	0.8600
PoterStemmer	0.3533	0.4243	0.8600
NO STEM	0.4200	0.3532	0.8524

Tab. 2: System performances with different stemmers

The different stemmers used are:

- EnglishMinimalStemmer<sup>6</sup>: Stemmer that simply stems plural English words to their singular form.
- Krovetz Stemmer<sup>7</sup>: Hybrid algorithmic-dictionary stemmer that produces words.
- Porter Stemmer<sup>8</sup>: Stemmer that eliminates the longest suffix possible, working by steps and trying to delete each suffix every time until it reaches the base form for generating stems.

As we can see from Tab. 2, the score obtained decreases by using stemmers, probably due to the limitations of stemmers used. So we decided not to use any stemmer.

## 3.2. Searcher

After setting up a basic function for search that simply uses a boolean query to check for matches in documents, we tried two different similarities and we tried to improve performances using various approaches: query boosting, query expansion and re-ranking. Our system produces essentially two ranks: one rank that contains the most relevant documents to the query and one rank that contains the most relevant sentences pairs. The second rank is obtained from the first one by analyzing the sentences contained in every document in the first rank.

### 3.2.1. Similarity decision

We tested the BM25 Similarity and the Dirichlet Similarity on the simplified version of the system, enhanced with no stop lists and no stemmers, which were the best solutions obtained from the previous tests. We tested this system with the two similarities, gaining the following results.

---

<sup>6</sup>[https://lucene.apache.org/core/4\\_10\\_2/analyzers-common/org/apache/lucene/analysis/en/EnglishMinimalStemFilter.html](https://lucene.apache.org/core/4_10_2/analyzers-common/org/apache/lucene/analysis/en/EnglishMinimalStemFilter.html)

<sup>7</sup>[https://lucene.apache.org/core/5\\_3\\_1/analyzers-common/org/apache/lucene/analysis/en/KStemFilter.html](https://lucene.apache.org/core/5_3_1/analyzers-common/org/apache/lucene/analysis/en/KStemFilter.html)

<sup>8</sup>[https://lucene.apache.org/core/8\\_0\\_0/analyzers-common/org/apache/lucene/analysis/en/PorterStemFilter.html](https://lucene.apache.org/core/8_0_0/analyzers-common/org/apache/lucene/analysis/en/PorterStemFilter.html)

Similarity	nDCG@5	P_5	RECALL
BM25	0.4428	0.5120	0.8436
Dirichlet $\mu = 1700$	0.3926	0.4720	0.8666
Dirichlet $\mu = 1800$	0.3870	0.4640	0.8663

Tab. 3: System performances with different similarities

After these results, we decided to use the BM25 similarity for our project. We did this test at the end of the project development, based on the results of the next sections about Query Boosting, Query Expansion and Re-ranking.

### 3.2.2. Query Boosting

Since documents have more than one field to search in, it is possible to assign different weights to each field at query time. In this way, a term found in a field with a higher weight will also have a higher impact on the final score of the document. As already explained in the section 3.1, we decided to index four main fields: Conclusion, Source Text, Discussion Title, and Source Title. So we decided to do query boosting with different weights on these fields and determine the best weight set to apply.

Source Text	Conclusion	Discussion Title	Source Title	nDCG@5	P <sub>5</sub>	RECALL
0.25	1	1	1	0.2974	0.3760	0.7473
1	0.75	0.5	0.5	0.3225	0.3920	0.7473
1	1	1	0.5	0.2974	0.3760	0.7492
2	1	1	1	0.3455	0.4280	0.7481
4	1	1	1	0.3410	0.4240	0.7391
2	2	1	1	0.3257	0.3920	0.7473
2	0.5	1	1	0.3457	0.4280	0.7410

Tab. 4: System performances with different fields weights

We noticed that Source Text is the most informative field; instead the conclusion, source title, and discussion title are often composed by a few terms, and very rarely these are relevant to the search. So, according to these considerations, the best score would be obtained by assigning a higher weight to the source text and a lower one to other fields, as we can also see from Tab. 4.

### 3.2.3. Query expansion

In order to increase the recall of the system, an idea was to implement query expansion. To do that we used:

- Rapid Automatic Keyword Extraction (RAKE) [15] algorithm
- Using Datamuse API [16]



The rake algorithm used is based on the implementation found in the Github repository [17]. This algorithm is used to recognize the most significant token of a query. Then synonyms of this token are retrieved using Datamuse API, this API returns for a given word a list of synonyms and a score for each one of them.

This process aims to generate queries that belong to topics similar to the original query. The results of both the original query and the generated ones are then mixed in a collection. If duplicates are found while merging the results list of the queries, the document with the highest score will be kept.

In the process of query expansion, we tried different approaches, and tuned some parameters: for example we considered as valid synonyms all the words which score is above half of the best synonym. Using this approach we were able to generate more than two hundred queries starting from the fifty we had originally. The runs in which we used this query expansion were worst (in terms of precision and recall) than the ones without query expansion. The best results we experimentally obtained were the one using only one synonym for the most significant token found using RAKE.

### 3.2.4. Re-Ranking

In order to increase our precision and our nDCG@5 we tried to re-rank documents. We tried to order the documents based on two metrics:

- Sentiment score of the document
- Readability of the document

An idea we had was trying to increase the scores of the document and sentiment was similar to the one calculated on the query. To do so, we used the VADER library: we calculate the sentiment of the query, then we boost the score of documents whose conclusion field has a similar value compared to the one of the query. To do so, we use a Gaussian distribution, and the score is boosted according to the interval in which the score of the document is. Experimentally we realized that the conclusion fields and the query fields are really short, so the boosting was not as significant as expected.

A similar story can be said for the readability score. Using the Flesch–Kincaid [18] metric, indeed the implementation found on GitHub [19] we ordered the documents based on their readability score. We tried to use it in the re-ranking since calculating the score of readability while indexing was very heavy computationally speaking. We decided to re-rank documents based on the readability computed on the conclusion field of the document. As occurred with the sentiment analysis before, we have that the score computed on the conclusion field is too unstable and not that significant due to the fact that the conclusions are extremely condensed.

## 4. Experimental Setup

Touch  offers us the possibilities to access the args.me corpus via the API of args.me search engine or downloading the file containing all the documents. We decided to download the entire corpus of the pre-processed dataset, that is available on the Touche Task 1 website.

## 4.1. Data description

### 4.1.1. Document

The updated version of the args.me corpus contains 365,408 arguments crawled from some debate portals such as debatewise.org, idebate.org, debatepedia.org, and debate.org. Each argument is identified by an ID, and it is constituted by a conclusion and one or more premises. Every argument is also associated with a stance (PRO or CON), some key sentences of the argument text, and also some information about the context like the source URL, the title of the discussion, and many others.

The collection can be found here:

<https://files.webis.de/data-in-progress/data-research/arguana/touche/touche22/>

### 4.1.2. Topics

For testing and tuning the system we used the 2021 Topics (only the title). Available at:

<https://webis.de/events/touche-22/topics-task-1-only-titles-2021.zip>

For the runs submission, we used the 2022 topics available at:

<https://files.webis.de/corpora/corpora-webis/corpus-touche-task1-22/topics.xml>

### 4.1.3. Relevance judgments

For testing and tuning the system we used the relevance judgments of the previous year, available at:

<https://webis.de/events/touche-22/touche-task1-51-100-relevance.qrels>

## 4.2. Evaluation Measures

We used three different evaluation measures to assess the quality of our system and of our different solutions. These measures were all calculated by the `trec_eval` executable, which was executed with the various test runs of the system:

- Normalized Discounted Cumulative Gain (nDCG) score with an evaluation depth of 5 (nDCG@5). In particular, we used the implementation provided by the `trec_eval` library to measure the performance of our IR system. The nDCG is the result of the equations Eq.1 and Eq.2 below. Parameter  $b$  indicates the patience of the user in scanning the result lists, and usually it is a value of 2 for an impatient user, or 10 for a patient user. Since the result is not bounded in  $[0,1]$ , it is necessary to normalize the score dividing nDCG by the Ideal Discounted Cumulated Gain (iDCG), provided by Touché Lab, as can be seen in Eq. 2

$$DCG@5 = \sum_{n=1}^5 \frac{relevance_n}{\max(1 + \log_b(n + 1))} \quad (1)$$

$$nDCG@5 = \frac{DCG@5}{iDCG@5} \quad (2)$$

- Precision at position 5 ( $P_5$ ), which indicates the percentage of how many relevant documents are in the first 5 positions:

$$P(5) = \frac{1}{5} \sum_{n=1}^5 r_n \quad (3)$$

In this case  $r$  is a binary relevance of the document ( $r=1$  document relevant,  $r=0$  document not relevant)

- Recall, for evaluating how many relevant documents were retrieved by the system in the whole rank:

$$R = \frac{|\text{relevant documents retrieved for that query}|}{|\text{relevant documents for that query}|} \quad (4)$$

$|\cdot|$  indicates the cardinality of the set, so the Recall is the ratio between the cardinality of the set of relevant documents retrieved for that query and the cardinality of the set of relevant documents for that query in the collection.

### 4.3. Repository organization

It is possible to find the source code of the project here:

<https://bitbucket.org/upd-dei-stud-prj/seupd2122-lgtm/src/master/>

The repository contains the following items:

- /code/
  - pom.xml -> maven file for the project dependencies
  - /src/main/java/
    - \* analyze/
    - \* index/
    - \* parse/
    - \* search/
    - \* utils/ -> support utils class and Main Class
  - /src/main/resources/ -> stoplists and stopwords
    - \* stoplists
    - \* stopwords

### 4.4. Hardware components

The devices used for parsing, indexing, searching and testing are:

- Model: MacBook Pro (13-inch, 2017)
- OS: macOS Big Sur version 11.6
- CPU: 2,3 GHz Intel Core i5 dual-core
- RAM: 8 GB 2133 MHz LPDDR3
- GPU: Intel Iris Plus Graphics 640 1536 MB
- Storage: SSD 256GB

- OS: Ubuntu LST 22.04 LTS
- CPU: i5 3570
- RAM: 8GB DDR3
- Storage: SATA SSD 256GB
- OS: Windows 10 PRO
- CPU: Interl i5 4670K
- RAM: 8GB DDR3
- GPU: Azootac NViDIA 780
- Storage: 1 TB HDD, 120 GB SSD

## 5. Results and Discussion

In the final phase of the project we tried to set the system in the best configuration by combining the various solutions that we developed in the search part. We obtained the following results:

Configuration	nDCG@5	P <sub>5</sub>	RECALL
Search with NO BOOST, NO QE, NO RE-R	0.3621	0.4320	0.7867
Search with BOOST, NO QE, NO RE-R	0.4428	0.5120	0.8436
Search with BOOST, QE in all token, NO RE-R	0.3961	0.4640	0.7527
Search with BOOST, QE only main token, NO RE-R	0.4294	0.5000	0.7925
Search with BOOST, QE only main token, RE-R Sent	0.2383	0.2760	0.6304
Search with BOOST, QE only main token, RE-R Read	0.1416	0.1800	0.4684

Tab. 5: System performances with different search solutions

In the previous table we use the following terms:

- "NO BOOST", "BOOST": to indicate if we are using query boosting (with the best weight set).
- "NO QE", to indicate that we are not using Query Expansion, "QE in all token", to indicate that we are performing query expansion in all the tokens returned by RAKE, "QE only main token", to indicate that we are performing Query Expansion only on the main token returned by RAKE.
- "NO RE-R" to indicate that we are not using Re-Ranking, "RE-R Sent", to indicate that we are performing re-ranking based on sentiment analysis on the conclusion field, "RE-R Read", to indicate that we are performing re-ranking based on the readability of the conclusion field.

After this final test, we conclude that the best configuration for our system is:

Configuration	nDCG@5	P <sub>5</sub>	RECALL
Search with BOOST, NO QE, NO RE-R	0.4428	0.5120	0.8436

The score achieved both by using synonyms to all tokens returned by rake and using synonyms only to the main token with the query expansion is lower than expected. This is probably caused by the fact that the expanded queries generated are sometimes satisfactory and sometimes unsatisfactory. So, many times they transform the original query to a worse one. Furthermore, by using more synonyms, we create a lot of noisy queries that degrade the results.

The sentiment analysis performed only on the conclusion field leads to a deficient solution, the same as the readability analysis. This is probably caused by the fact that these two scores have to be calculated on the source text of the document. We couldn't do this test because the sentiment and readability re-ranking based on source text requires a lot of time in the search process. Finally, the BM25 Similarity seems to work better than the LMDirichlet, unless it seems the last one worked better in the previous year's studies.

## 6. Conclusions and Future Work

Over the last decade, search engine quality has improved dramatically due to the uninterrupted optimization of information retrieval effectiveness. Furthermore, Argument retrieval for controversial questions is one of the fields that can significantly impact the latest information retrieval systems. Moreover, to be involved in the improvement of this area, we participated in Touché 2022 Lab task 1 proposed by CLEF[2] which emphasizes retrieving and ranking sentences that carry consequential points related to the controversial topic.

We implemented our IR system to retrieve the most relevant sentence pairs to the given queries provided in the Touché shared task. We tried to test different solutions in terms of stoplists, stemmers, and similarities. In addition, we created the system based on the best results.

We also showed how essential it is to give the right weight to the different parts of a document, since a lot of information can be useless during the search.

Nevertheless, there are some aspects that can be improved to reach better performances. For example, in query expansion, it can be helpful to use some NLP tools that can rewrite one query to another one that is related to the first one in a meaningful way. Furthermore a pseudo relevance feedback technique can also be added in the query expansion phase.

Another improvement can be made by using a better formula to re-rank the documents or using a different score instead of the one formulated with sentiment and readability analysis. For example, with a machine learning approach, it would be possible to train a model to assign a quality score to each topic and then use this value to re-rank the top retrieved documents.

As a final observation, we presented our approach to the problem, and we think that in the future always better solutions will be presented, especially with the help of machine learning and high computational platforms.

## References

- [1] C. D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, Cambridge, UK, 2008. URL: <http://nlp.stanford.edu/IR-book/information-retrieval-book.html>.
- [2] CLEF, Touche', 2021. URL: <https://webis.de/events/touche-21/>.

- [3] University of Padova - Search engines course, Department of information engineering, Student projects of search engines course, <https://bitbucket.org/upd-dei-stud-prj/workspace/projects/SE2021/>, 2021.
- [4] A. S. Foundation, Apache Lucene, 2022. URL: <https://lucene.apache.org/core/corenews.html#apache-lucenetm-910-available>.
- [5] Apache Software Foundation, Apache lucene similarities, [https://lucene.apache.org/core/7\\_0\\_1/core/org/apache/lucene/search/similarities/Similarity.html](https://lucene.apache.org/core/7_0_1/core/org/apache/lucene/search/similarities/Similarity.html), 2022.
- [6] Wikipedia contributors, Analysis of variance — Wikipedia, the free encyclopedia, [https://en.wikipedia.org/w/index.php?title=Analysis\\_of\\_variance&oldid=1085080872](https://en.wikipedia.org/w/index.php?title=Analysis_of_variance&oldid=1085080872), 2022.
- [7] T. community, Trec eval, 2008. URL: [https://github.com/usnistgov/trec\\_eval#readme](https://github.com/usnistgov/trec_eval#readme).
- [8] Princeton University, Princeton university 'about wordnet.', <https://wordnet.princeton.edu/>, 2010.
- [9] Apache Software Foundation, English minimal stemmer, [https://lucene.apache.org/core/4\\_10\\_2/analyzers-common/org/apache/lucene/analysis/en/EnglishMinimalStemmer.html](https://lucene.apache.org/core/4_10_2/analyzers-common/org/apache/lucene/analysis/en/EnglishMinimalStemmer.html), 2021.
- [10] Alias-i, Lingpipe 4.1.0, <http://alias-i.com/lingpipe>, 2008.
- [11] T. A. O. library Introduction, The Apache OpenNLP library , 2022. URL: <https://opennlp.apache.org/docs/1.9.4/manual/opennlp.html#opennlp>.
- [12] C. Hutto, E. Gilbert, Vader: A parsimonious rule-based model for sentiment analysis of social media text, in: Proceedings of the international AAAI conference on web and social media, volume 8, 2014, pp. 216–225.
- [13] N. Ferro, Hello Tipster repository, 2021. URL: <https://bitbucket.org/frncl/se-unipd/src/master/hello-tipster/>.
- [14] Jackson Contributors, Jackson project home, <https://github.com/FasterXML/jackson>, 2021.
- [15] S. Rose, D. Engel, N. Cramer, W. Cowley, Automatic Keyword Extraction from Individual Documents, 2010, pp. 1 – 20. doi:10.1002/9780470689646.ch1.
- [16] Datamuse API, Datamuse, <https://www.datamuse.com/api/>, 2022.
- [17] Jackson Contributors, Rake, <https://github.com/aneesha/RAKE>, 2015.
- [18] Wikipedia contributors, Flesch–kincaid readability tests — Wikipedia, the free encyclopedia, 2022. URL: [https://en.wikipedia.org/w/index.php?title=Flesch%E2%80%93Kincaid\\_readability\\_tests&oldid=1082199491](https://en.wikipedia.org/w/index.php?title=Flesch%E2%80%93Kincaid_readability_tests&oldid=1082199491).
- [19] whelk.io, Flesch-kincaid java implementation, <https://github.com/whelk-io/flesch-kincaid>, 2022.